TENABLE
Network Security®

Nessus 3.0
Advanced User Guide

August 1, 2006
(Revision 7)

The newest version of this document is available at the following URL:
http://www.nessus.org/documentation/nessus_3.0_advanced_user_guide.pdf

Table of Contents

Note: Currently Tenable Network Security, Inc. is going through a name change process for all of our products. The directories, commands, configuration files, etc. will still reflect the old names for the Log Correlation Engine and Passive Vulnerability Scanner until new versions are released. Tenable would like to thank you for your patience through this process.

Lightning will be known as **Security Center**
Thunder will be known as **Log Correlation Engine**
NeVO will be known as **Passive Vulnerability Scanner**

Introduction

Welcome to Tenable Network Security's **Nessus 3.0** Advanced User Guide. As you read this document, please share your comments and suggestions with us by emailing them to support@tenablesecurity.com.

Tenable Network Security, Inc. is the author and manager of the Nessus Security Scanner which is offered to the world as free software. In addition to constantly improving the Nessus engine, Tenable is in charge of writing most of the plugins available to the scanner. Nessus has been widely deployed worldwide on Windows, UNIX, and OS X operating systems. While Nessus has become a staple for many organizations, there are still many features of Nessus which are not often utilized. The purpose of this document is to elaborate on some of Nessus' "dustier corners". This document will also quickly cover the new features within Nessus 3.

What's New in Nessus 3

Nessus 3 is the latest version of Nessus. Tenable Network Security, Inc. (Tenable) offers Nessus 3 as a **free** product for UNIX, Windows, and OS X operating systems. A non-exhaustive list of changes between Nessus 2 and Nessus 3 includes:

- NASL3 is 16 times faster than NASL2 and a full 256 times as fast as NASL1.
- The IDS-evasion feature is gone.
- Nessus 3 has more protocol APIs (see http://www.tenablesecurity.com/images/pdfs/reliability_and_uniqueness_of_nessus.pdf for more information).
- In Nessus 3, each host is tested in its own individual process. Scripts share the same process space; however, due to the VM implementation of Nessus 3, there is perfect containment of scripts.
- A NASL script can only use 80 Megs of memory. This is a very large value and, in fact, most scripts use less than 512K.
- The NASL3 VM is very secure. A poorly written NASL script is not vulnerable to any buffer/stack overflows or memory corruption because the language itself prevents that kind of problem from occurring.
- There are two kinds of NASL functions. Namely:
    - o "Harmless" functions which **cannot** interact with the local system (i.e. Open sockets, execute local commands outside the sandbox, etc.).
    - o Functions which can interact with the local system are supported in Nessus 3; however, the script must be signed by Tenable. In this way, tainted scripts cannot interact with the local system and the risk of a script being copied from system-to-system is reduced.
- With Nessus 3, there are over 10,000 vulnerability checks available.
- Compliance scans (see below for more information).

Scanning Modes and Nessus OS fingerprinting

"What's in a name? That which we call a rose by any other name would smell as sweet"
--Shakespeare

The ability to detect the operating system of a remote target is critical. While some systems will voluntarily give up their version information, sometimes a bit more conjuring is required. A vulnerability scanner must be able to adapt to many different environments. That is, the scanner cannot assume anything regarding the host that it is scanning. One of the initial steps that Nessus takes is to attempt to identify the remote operating system. This is a highly critical step, as the other Nessus modules will often rely on this information to make intelligent decisions about whether or not to scan the target host.

Nessus utilizes the following means to determine the operating system:

- If the system appears to be a Windows system and Nessus can access the registry, then Nessus will simply pull the OS version from the registry directly.
- SNMP – if the system can be queried for its system information, then Nessus uses this information.
- ICMP – Nessus sends malformed ICMP requests to the system (as outlined in Ofir Arkin's paper, ICMP Usage in Scanning).
- If the remote host is a Windows host, it will attempt to determine its OS type by sending MSRPC packets to port 139 or 445 and guess the OS based on the results.
- If the remote host has an NTP client listening on port 123, Nessus will try to ascertain the OS through NTP queries.

If the above checks are run and the operating system is still not known, it may be for a number of reasons. These may include:

- The host may be firewalled. If there are no open ports, or if the system filters certain ICMP types, then Nessus may not be able to obtain an accurate reading on the Operating System.
- The TCP/IP stack on the remote host may have been purposefully obfuscated.
- The IP address being tested may be a load-balancer which is spreading the traffic across multiple backend systems.
- There may be a Port Address Translation (PAT) server between the scanner and the scanned host.
- The remote host may be running a non-RFC compliant application.
- The remote host may be so obscure that no one has submitted any fingerprints for such a host.

Why is Nessus fingerprinting safe for an organization to use?

The question often arises as to whether Nessus OS fingerprinting is safe. Nessus OS fingerprinting is very safe. Anytime you scan a system, there is a risk of doing harm to the system; however, the Nessus checks are engineered to be as safe as possible. That is, extensive testing (both by Tenable and by many thousands of current users) has shown that the Nessus OS fingerprinting should not harm network machines.

Rapid Host Discovery Scanning

Many organizations will not be able to run full scans against their entire enterprise on a routine basis. For these organizations, host discovery scanning will give them information regarding dispersal and types of resources. A Rapid Host Discovery scan utilizes only a few Nessus modules and quickly sweeps the target network for hosts. A scan administrator can choose many different methods of rapid host discovery scanning. As an example:

- ICMP (or ping) scanning will quickly return a list of "typical" workstations and servers. Note that systems which have disabled ICMP or are firewalled will not show up in such a scan.
- TCP quick port scanning means scanning a network quickly with a TCP scan of certain port ranges. For example, most Windows systems have port 139 or 445 open. Most web servers listen on port 80. Many UNIX systems have port 22 or 23 open for remote administration. A scan administrator can configure a TCP scan to only sweep certain ports. These scans will return very quickly and give the administrator a good idea what resides on the network.

These are just a few suggestions for rapid host scanning. Each organization will want to plan and implement a rapid scan with input from administrators, business units, and other IT units. When properly implemented, a rapid scan can very quickly give an organization information regarding resource dispersal.

Slow, Complete Scanning

Where a rapid scan gives quick, generic information regarding hosts on the network, complete scans can take quite some time on large networks. Complete scans are, of course, needed on a regular basis. Tenable releases dozens of new vulnerability checks each month. Systems and configurations can change rapidly. A complete Nessus scan ensures that systems are not vulnerable to the newest vulnerabilities. Complete scans begin with a quick network sweep, an OS fingerprint, and then, based on the results, run a complete battery of relevant checks against the target host. Running a complete scan is more than just selecting all the checks. There are some issues that should be taken into consideration. Namely:

- Brute Force attacks – A brute force attack attempts to gain access to a resource through many attempted logins. Brute force scans, when properly planned, can give an organization a benchmark of account and password integrity.
- When to Port Scan all UDP ports? A UDP Port scan takes a considerable amount of time and can sometimes be prone to false positives. UDP port scanners, in general, send a packet and await an ICMP packet if the UDP port is not in use. Firewalled hosts or ICMP filtering (anywhere between the Nessus scanner and the target host) will cause the port to **appear** open. UDP port scans should typically be limited to a specific target range. If Nessus reports that nearly all UDP ports are open, this typically indicates that the host is firewalled or ICMP packets are being filtered somewhere.

Inherent Dangers of Complete Scans

There are inherent dangers with running a complete scan. A non-exhaustive list of potential problems would include:

- There are still many legacy operating systems which cannot handle many simultaneous open sockets. A port scan sometimes causes these systems to lock up.
- Systems which have implemented an "Account Lockout" policy will sometimes deny services to valid users if the Nessus scan has locked out an account due to multiple failed logins.
- Denial of Service (DoS). We deal with this subject in more detail below.
- Network degradation. A Nessus scan can be intensive on the network.

Speed, Accuracy, and Stability

When planning a complete scan, you will be interested in tradeoffs between speed, accuracy, and stability. If you scan too fast, you risk degrading the network bandwidth or losing packets. If you choose to scan fast during non-business hours, you risk not seeing the machines which may be turned off (less accuracy). If you wish to maximize accuracy, then stability and speed will suffer, etc. That is, by attempting to maximize any of the three areas above, you will cause the others to decrease. There is not one magic formula for each organization. As a security administrator, you will want to meet with business clients and come up with a suitable plan for addressing these needs.

Deploying a Nessus Infrastructure

"Know thy network"

Prior to deploying a Nessus infrastructure, you should understand your target network. For example:

- Where are the network bottlenecks?
- Where does one network end and another (business partner or otherwise) begin?
- Where are the firewalls?
- Where are your RFC 1918 networks?
- What routing protocols are used?
- What network protocols are typically used?
- Where are the legacy mainframes or other similar equipment?
- Where are the critical network infrastructure machines (Authentication servers, Primary Domain Controllers, etc.)? For example, you are looking for any machine which, if downed, would kill connectivity for many clients. If you begin your scan by crashing a next-hop router, scan results will be horribly skewed.

Too many "green" security administrators just crank up Nessus, select all the plugins, and scan an entire corporate network. This is foolhardy (to say the least, negligent to say the most). A vulnerability scan is an offensive entry into a network. Care should be taken to understand the network and work with the system owners **prior** to scanning a network.

Speed, Speed, Speed!

Nessus 3 is very fast. With Nessus 3, the network is the limiting factor. Given this, if more speed is required, you will want to have multiple Nessus engines running in parallel. Ideally, you would have a separate Nessus engine for each separate network segment (VLAN, Class C, broadcast domain, etc.). Besides speed, you will also get other benefits from such a configuration. For example, when scanning a local broadcast domain, your Nessus scanner will be able to pick up on things which typically would not be routed beyond the next-hop router. By having a scanner on each broadcast domain, you can detect and use broadcast traffic, RFC 1918 addressing, etc. Having separate scanners ensures that the Nessus scanning traffic does not traverse WAN pipes.

Tenable offers the Security Center (formerly Lightning Console) to manage multiple Nessus scanners. Nessus 3 runs on UNIX, Windows, and OS X operating systems. An organization can, for example, deploy the Windows version of Nessus on their Backup Domain Controllers (or other local machines which typically have spare memory and CPU cycles). Since each scanner is only responsible for a small portion of their local network, it is not always necessary to have a **dedicated** machine for each of these remote scanners. The Security Center can then manage all of these scanners. Such a configuration could scan large address ranges (multiple Class B networks) in a matter of hours.

Location, Location, Location!

It is very important to plan on where a scan should originate. Do you want to simulate the "hackers" view and scan from outside the network? Do you want to scan from inside a network? Do you want to scan from a business partner network into your network? There are hundreds of permutations. The first question will be: "which vector of attack do I wish to test for?" Ideally, you want to test all the different permutations. A strong Internet with a weak Intranet spells trouble for any organization. A few common scanning location configurations include:

- Scanning from inside the network against critical machines. If your organization has a Disaster Recovery Plan (DRP), this list of critical machines should already be available.
- Scanning from outside the network against Internet-facing machines. This is very common and most organizations probably already do this.
- Scanning Internet-facing machines from a scanner on the same broadcast domain. What happens if a web server is compromised? Do the rest of the Internet-facing machines fall over like dominoes? Internet-facing machines should be hardened. Period. Assume that an attacker will compromise an Internet-facing machine and scan accordingly.
- Scanning from a business partner network into your corporate network. This can be used as a Compliance scan after (or before) a firewall audit. Security is only as strong as its weakest link. A trust relationship with an unsecured business partner can be a deadly error.
- Scanning from inside the network against all internal machines. Unfortunately, this is often the first scan that is done by security administrators. This can lead to such a glut of information that months are spent prioritizing Risk and assigning work to stunned, already overworked administrators. This is useful if you have already hardened Mission-Critical systems.

Time, Time, Time!

How often do you scan? There is no simple answer. If active scanning is the only scanning being done, then you will wish to scan as often as possible. Most organizations utilize Change Control Procedures. Try to scan **after** a change control window. Remember, if you are scanning every 30 days, a change in the network 2 days after a scan will go undetected for 28 days. While outside the scope of this paper, Tenable offers a 24x7 passive vulnerability scanner which detects these changes in real time [1]. With respect to **time**, Tenable releases dozens of plugins per month. Be sure to have your Nessus scanner set up to automatically retrieve the latest direct feed from Tenable [2] prior to a scan. When arranging scans, prioritize based on the criticality of the system being scanned. It may not make sense to scan the receptionist's computer once a month, but you may want to scan your Internet-facing machines twice monthly.

Dustier Corners

Application Scanning

The administrator who just enables everything and hits the "Start" button is missing a good opportunity to find some relevant flaws. If you are scanning web servers, you will want to ensure that "CGI scanning" is not disabled within the Nessus configuration.

Within the Nessus configuration file, there is a default "timeout" value for each script. At times, a security administrator may wish to adjust these values. If you are scanning a host that is 40 hops away, the default timeout value may be inadequate. If you "know your network" (see above), then you can plan how to adjust this value accordingly.

Within the *webmirror.nasl* script, it explicitly states:

"*This script makes a mirror of the remote web site(s)
and extracts the list of CGIs that are used by the remote
host.*

*It is suggested you give a high timeout value to
this plugin and that you change the number of
pages to mirror in the 'Options' section of
the client*."

Corporations are moving more and more content to their web servers. If you do not follow the advice above, you are putting yourself at risk. It is not uncommon to routinely audit web servers which have more than 20,000 unique content pages. Many other NASL scripts rely on *webmirror.nasl* to find and report on found directories, Cross-Site-Scripting (XSS) flaws, and more. One of these scripts (*sql_injection.nasl*) simply cannot be effective without a full listing of CGI programs residing on a host. Talk with the application development teams for a rough estimate on the number of unique content pages, then increase your default Nessus "number of pages to mirror" to 120% of the estimated number. Adjust your timeout value accordingly. You do not want to be the security administrator who missed a serious flaw just because you did not give your scanner enough time to do its job.

The detection of Application-layer flaws within HTTP (or web) applications has become a major source of activity for many enterprise security groups. Source code audits are one means of finding these security holes. Many of these efforts to detect application bugs within web applications can be automated by using the Nessus scanner. This section will discuss the techniques used by Nessus to efficiently scan for application layer bugs. Familiarity with Nessus, HTTP, SQL, and Active Scripting (CGI) is assumed.

Why should an organization put an emphasis on HTTP applications?

A Web application bug can be every bit as devastating as a network-level bug. To further exacerbate the problem, many companies and institutions **allow** web traffic to/from their secured network segments (DMZ). An attacker that gains access to a web server may have the ability to modify or create web content. The attacker may also gain access to business-critical data (such as business logic, passwords, etc.). Furthermore, the attackers can use their existing access to "escalate" their privileges on the internal network. While the firewall may block access from the Internet into the internal network, most firewalls must, necessarily, allow their web servers to communicate with internal machines and other DMZ machines.

NOTE: *Nessus checks for many web/CGI vulnerabilities (literally, hundreds). The scope of this section is the detection of "unknown" or "undocumented" vulnerabilities within a web* ***application****. That is, the methodologies denoted below will find inherent weaknesses within "homegrown" or "custom" web applications. This is significant in that it is a departure from "signature based" vulnerability scanning.*

The plugin which mirrors the website is *webmirror.nasl* with a Plugin ID of 10662. *Webmirror.nasl* does more than just "mirror" the website and populate the knowledge base. *Webmirror.nasl* also looks for inherent weaknesses within the active scripts (or CGI scripts) that it finds. Furthermore, as *webmirror.nasl* runs, it finds all active (or CGI) scripts. Nessus notes the location of these scripts and ferrets away the information for further testing. So, for example, if there was a link to a file called */my-secretcgi/foo.cgi*, Nessus would make note of the directory */my-secret-cgi* and include this directory in **all** future CGI directory checks.

What about directories that are not linked via HTML, but still exist?

A NASL module, *DDI_Directory_Scanner.nasl*, will find directories (both visible and "hidden"). It does this by using a brute-force technique against the web server. So, for example, if the web administrator created a directory named "backup" which he/she uses to store old password files, *DDI_Directory_Scanner.nasl* would find this directory and report on it. This NASL module will find directories which are not "linked" to the actual web server content. As many administrators rely on security through obfuscation, this NASL module is a critical piece of any web application penetration test.

Another NASL module, *torturecgis.nasl*, takes the output from *webmirror.nasl* and tests each of the active scripts (or CGI scripts). *Torturecgis.nasl* includes checks for:

- Cross-site-scripting (XSS). XSS allows a malicious external user to potentially run active code against unsuspecting browsers (or email clients) within the security scope that is applied to the target web server. An example is probably in order. Consider a site which has a form that is vulnerable to XSS. A malicious individual can craft a URL that looks valid (http://www.vulnerable.site/vulnerable.asp?myinfo=<script>...</script>). Now, in this example, everything between the <script> and </script> would be executed on the client browser with the security rights of the site www.vulnerable.site. Of course, this could be used to retrieve and forward cached credentials, steal cookies, besmirch company image, etc.
- Directory escape techniques. For example, a "../" within a directory path could force the web server to access files outside of the web directories. A common example is the use of "../../../../../etc/passwd" to obtain the UNIX password file.
- Hidden form values. Many sloppy applications will pass or store credentials as a "hidden" value.

Another NASL module, *sql_injection.nasl*, takes the output from *webmirror.nasl* and performs SQL injection and blind SQL injection checks.

What is SQL Injection?

SQL injection is an Application layer (or layer 7) attack. At this time, it is one of the more popular attack methods employed by professional penetration testers as well as hackers. SQL injection attacks exploit flaws that are caused by the failure to properly validate input data by the programmer, and are a common root cause of many of the popular Application layer attacks.

Within SQL statements, certain characters have a special meaning. For instance, the semicolon tells the SQL engine to stop processing the statement, and the keyword OR changes the logic of the SQL statement. When an application fails to strip out one of these special characters or strings, the logic of the SQL statement can be compromised. This can lead to unauthorized system or data access. SQL injection attacks commonly use a "front-end" (or ingress) machine as their attack vector. Typically web sites are connected to a back-end SQL server. The attacker typically tests the web server for CGI scripts or forms that return error codes regarding SQL. Once an attacker finds a script or form which does not validate the user input, it is a short time before the attacker gains full access to the SQL server.

You can test for the SQL injection manually; however, this can be very time consuming. To manually check for SQL injection flaws (not recommended), you must first identify the ingress point of the SQL statement (i.e., the Web form, CGI, etc.) and map out each of the input parameters. For example, www.thisNEwidget.com/orderform.asp is a Web form which accepts four input variables (text boxes named box, box2, box3, and box4). A typical browser address bar may look like:

```
http://www.thisNEwidget.com/orderform.asp?box=Joe%20Smith&box2=Atlanta
%20GA&box3=widget%20order&box4=654333
```

(Note that "%20" is just how the browser represents a space.)

In this imaginary order, a user named Joe Smith from Atlanta, Georgia, is placing a widget order for item number 654333. In the browser window, you can tamper with the values of the four variables. By changing the URL to (note the trailing semicolon):

```
http://www.thisNEwidget.com/orderform.asp?box=Joe%20Smith&box2=Atlanta
%20GA&box3=widget%20order&box4=;
```

You may receive a SQL or database error. The site is probably vulnerable to SQL injection if this occurs. Once a SQL injection flaw is found, it is typically manually exploited. Common exploit methods include calling a stored procedure (such as `xp_cmdshell`, a Microsoft SQL default stored procedure), creating a SQL query which dumps all table data to an HTML table, and copying a file (like *cmd.exe*) into the web root directory.

How Does Nessus Detect SQL Injection Vulnerabilities?

Nessus automates the process denoted above. *Sql_injection.nasl* takes the output from *webmirror.nasl* and queries each form using an invalid SQL statement. Nessus then looks at the reply from the web server. So, for example, if the web server returned an error which contained the string "You have an error in your SQL syntax", Nessus would flag the application as being vulnerable to SQL injection.

Blind SQL Injection is a vulnerability that has all of the same ramifications of a standard SQL injection attack; however, the site has taken the time to remove any SQL-specific errors returned by the server. So, a malformed query will **not** return a SQL error. To test for this sort of situation, Nessus sends a valid query with a valid SQL statement appended. An example is required. Consider the application which takes a form field (VALUE1) and puts it into a SQL query like:

```
SELECT * FROM table1 WHERE VALUE1 = 7
```

If we change “VALUE1” to:

```
7 AND 1=1
```

Then, the SQL query becomes:

```
SELECT * FROM table1 where VALUE1 = 7 and 1=1
```

That is valid SQL, but should be an invalid value with respect to the form. If such a query does not generate a different HTML page as opposed to the valid query, then Nessus flags it as being vulnerable to a Blind SQL Injection attack. Much more can be said regarding Blind SQL Injection, but that is left as an exercise to the reader.

Scanning With Credentials

If you are part of a security group within an organization and have worked with the other Information Technology groups **prior** to scanning, then you may have a good chance of obtaining credentials for a local scan. With credentials, Nessus can log into remote hosts and check the local configuration. This can be very beneficial as Nessus cannot routinely determine the version numbers of browsers, chat clients, anti-virus clients, etc. Scanning with credentials can save the administrator a lot of time in tracking versions of locally installed software as well as software (such as spyware) which should not be installed in the first place.

Nessus has the ability to log into both UNIX and Windows machines using SSH. This means that Nessus has the ability to use the **local** system to query itself for version information. With Windows, this means reading the hard drive, registry, or even executing commands. With UNIX, it is very similar, taking into account the different package managers that come with the different UNIX versions. See [4] for more information.

As of this writing, there are dozens of government regulations (required) and standards (guidelines) which apply to most major organizations. As such, policy compliance scanners have become popular. These are specialized scanners which only look for a subset of vulnerabilities in order to give some baseline level of Regulatory compliance. Nessus 3 has this built into the core product. In Nessus 3, you simply create a template file denoting which items (from the regulations or policies) you wish to check for. Nessus then automatically creates a list of relevant modules which it runs sequentially, giving a compliance report when the scan is complete.

Making the Knowledge Base (KB) Work for You

Most Nessus users are aware that Nessus stores information in a Knowledge Base. However, not many users are aware of the simplicity of searching the KB for information. If an organization is writing custom checks, they should be aware of the underlying KB format, as well as how to access it via the KB API.

As an example from *ssh_get_info.nasl*, we see a line which looks like:

```
set_kb_item(name:”Host/RedHat/release”, value:buf);
```

This is an example of setting a KB item. If you are writing a NASL which determines the version of a remote Red Hat installation, you would want to start with the following:

```
script_dependencies(“ssh_get_info.nasl”);
```

This line instructs Nessus to run *ssh_get_info.nasl* **before** running the current script. Within the script, you would want a second line which looked like:

```
get_kb_item(“Host/RedHat/release”);
```

This line retrieves the KB item that was dropped off by the initial script.

Compliance Checks

Governmental and Industry Regulations and Standards have proliferated in the last 5 years. Many organizations are required to be compliant with these regulations. Failure to be compliant can lead to severe business impact (loss of business, fines, and even jail time in some instances). Tenable realizes the need for companies to use Nessus as a Compliance or Audit tool. That is, once a company has taken the Regulations and Standards and created a policy specific to their business, Nessus can be used to test for the level of compliance. As each organization will have a unique policy, Tenable is creating a means of allowing the security administrators to create their own policy template files which are then **read** by Nessus 3. This means that administrators no longer need to modify NASL source code or “scan templates” in order to create a compliance scan. Changes to a policy can be reflected in one central configuration file.

The scan is done. What now? Handling Scan Results.

Dealing with False Positives

René Daumal is quoted as having said, “Truth is one, but error proliferates. Man tracks it down and cuts it up into little pieces hoping to turn it into grains of truth. But the ultimate atom will always essentially be an error, a miscalculation.”

Nessus plugin writers go to great lengths to **anticipate** how a machine will respond to a given query. However, you cannot anticipate every permutation. At some point, every plugin is vulnerable to a false positive. A false positive is where the plugin determines that a system is vulnerable when, in fact, it is not actually vulnerable.

Examples of false positives might include:

- Scanning a machine which is actually forwarding all traffic to a backend system (Network Address Translation)
- Scanning a machine which is actually forwarding multiple ports to multiple backend systems (Port Address Translation)
- Scanning a machine which is actually a virtual address for multiple machines.
- Scanning a machine which has been purposely altered to give the scanner a false read.
- Scanning a machine which is proxying TCP connections (such as the OpenBSD “scrub” option)

Troubleshooting False Positives

If you followed the advice above, then false positives will often just jump off the page at you. If you "know your network", then you will know that an Apache web server should not be vulnerable to a Microsoft WEBDAV overflow.

The following are some steps that can be taken when you suspect a false positive:

- display() is your friend. Edit the NASL script and put a display() call every few lines. Display responses from the server. Display the request that prompted such a response. Example: `r = recv(socket:soc, length:65535);  display(r);`
- A sniffer is your friend. Use your sniffer to read and record all traffic between the Nessus scanner and the target IP. Ideally, you will want to only trap the traffic while the one specific NASL script is running. Examine the stimulus sent to the server and the response. Compare this information with the actual NASL code.

Letting Tenable Network Security Troubleshoot Your Issues

Tenable is always interested in improving Nessus and reducing false positives. In fact, we sometimes reward frequent bug submitters with free Tenable plugin feeds! Here is what you will need if submitting a false positive report to Tenable:

- Send your tracefile. Tcpdump or Ethereal tracefiles are required (pcap format). You can generate a tcpdump tracefile by running something like: `tcpdump -i <interface> -s0 -w for_tenable.trace <pcap filter>`
  Note: With the Nessus 3 Windows client and/or nessuswx1.4.6, you will be able to instruct Nessus to gather the pcap capture.
- Send screen shots, where applicable.
- Send relevant KB items.
- Telnet to the port and report back banners where applicable.
- If applicable, send the HTML code that is generating a false positive.
- Retrieve relevant messages from nessusd.messages.

Minimizing Crashes

There is an inherent risk with any security scanner of crashing or harming remote systems. Many organizations still have many legacy networks, operating systems, and applications running. Nessus begins most scans with a probe of availability, service ports, etc. This, in and of itself, is sometimes sufficient to crash older systems and networks.

Organizations which are still very reliant on legacy systems or have older implementations which cannot suffer outage (SCADA, Process Control Networks, etc.), Tenable's Passive Vulnerability Scanner (formerly NeVO) may be required until the functionality of the systems can be moved to a more hardy environment [1].

A network may crash for the following reasons:

- Firewall has been crashed by the scanner
- Router or Switch has been crashed by the scanner
- Scanner has been auto-blocked by a firewall or IPS

If a scan returns too quickly or with very few results, you may wish to check that the target network is still available. There are a number of tools such as ping, sing, traceroute, etc. which can show you where your traffic may be getting lost.

What Makes Systems Crash?

While not nearly exhaustive, the following are a few reasons that a system may crash:

- The remote device may have trouble maintaining STATE connections and may crash when too many connections are made to the different service ports.
- The host may be alive, but the service is dead. In this case, you may still be able to ping the host or reach other services.
- The host may be alive along with the service; however, the service may be unresponsive. This sometimes happens when the service hangs while not closing incoming socket. If you can reach the host and telnet to the port, but get no response from the application, then this may be the case.
- The host is alive, but later dies. This happens rarely but there are documented cases of where an application may survive the scan just fine but crash when the service writes some information to a log file or accesses some memory that is corrupted.
- Sometimes, many servers will share a single back-end server (SQL server, COM server, etc.). By scanning multiple front-end servers simultaneously, you may be amplifying traffic to a back-end server, which causes a service interruption.
- Poorly written applications often can not handle port scans or simple HTTP GET commands.
- Some applications will crash when too many open sockets are requesting information simultaneously.
- Some applications will stop or crash when the log files get full.
- Some applications will stop or crash when the temporary directory gets full.

For Further Information

Tenable hopes your experience with Nessus is very positive, and we strongly encourage you to contact us via email or phone to discuss any issues you have. Tenable has produced a variety of other documents detailing Nessus' installation, deployment, configuration, user operation, and overall testing. These are listed here:

- **Nessus Installation Guide** – step by step walk through of installation
- **Nessus Client Guide** – how to install, configure, and operate the various clients available for Nessus
- **Nessus Credential Checks for UNIX and Windows –** information on how to perform authenticated network scans with the Nessus vulnerability scanner
- **Real-Time Compliance Monitoring** – outlines how Tenable's solutions can be used to assist in meeting many different types of government and financial regulations

Please feel free to contact us at support@tenablesecurity.com, sales@tenablesecurity.com or visit our web site at http://www.tenablesecurity.com. For more information about Nessus, please visit http://www.nessus.org.

References

[1] http://www.tenablesecurity.com/products/pvs.shtml
[2] http://www.tenablesecurity.com/buy/index.shtml
[3] http://www.tenablesecurity.com/images/pdfs/sec_test_sc3_nessus.pdf
[4] http://www.tenablesecurity.com/images/pdfs/blended_security_checks.pdf

About Tenable Network Security

Tenable, located in Columbia, Md., develops enterprise security solutions that provide vulnerability management, intrusion detection, and security event notifications across entire organizations for effective network security management. Tenable is uniquely positioned to detect vulnerabilities with active and passive scanning and analysis, and host-based patch monitoring for enterprise networks. Key product lines include: Nessus Vulnerability Scanner, the leading global technology utilized for vulnerability scanning; Passive Vulnerability Scanner (formerly NeVO), for passive vulnerability monitoring; Security Center (formerly Lightning Console), for enterprise security management; and Log Correlation Engine (formerly Thunder), for secure log aggregation and analysis. For more information, please visit us at http://www.tenablesecurity.com.

TENABLE Network Security, Inc.
8830 Stanford Blvd.
Suite 312
Columbia, MD 21045
TEL: 1-877-448-0489
http://www.tenablesecurity.com